Universität Bremen
FB 3 – Informatik
Prof. Dr. Rainer Koschke
TutorIn: Euer/Eure TutorIn

Software–Projekt 1 2013

VAK 03-BA-901.02

Architekturbeschreibung

Alexander Mues	mues@tzi.de	2789057
Hubert M. Estenfelder	estenfhu@tzi.de	2906702
Tamim Wahage	twahage@tzi.de	2588919

Abgabe: 27.Juni 2013 — Version 1.5

Inhaltsverzeichnis

Version und Änderungsgeschichte

Version	Datum	Änderungen
1.0	5.06.2013	Dokumentvorlage als initiale Fassung kopiert
1.1	12.06.2013	Einflussfaktoren und Problemkarten sind hinzugefügt worden
1.2	19.06.2013	Konzeptionelle Sicht ist hinzugefügt worden
1.3	24.06.2013	Modulsicht ist hinzugefügt worden
1.4	26.06.2013	Datensicht und Anwendungsfall sind hinzugefügt worden
1.5	27.06.2013	überarbeitete Abgabeversion

1 Einführung

1.1 Zweck

Entfällt in SWP-1

1.2 Status

Entfällt in SWP-1

1.3 Definitionen, Akronyme und Abkürzungen

1.4 Referenzen

1.5 Übersicht über das Dokument

Entfällt in SWP-1

2 Globale Analyse

2.1 Einflussfaktoren

2.1.1 Organisatorische Faktoren

author: Alex Mues

O1. Time-To-Market

Auswirkungen

- Das Produkt muss vollständig zum Abgabedatum fertiggestellt sein.

Flexibilität

- Keine Flexibilität. Wir können über das Abgabedatum nicht verhandeln.

Veränderlichkeit

- Eine Veränderung des Abgabe-Termins ist nicht warscheinlich.

O2. Budget

Auswirkungen

- Kein Budget vorgegeben. Kauf von Software und Bezahlen von Experten ist nicht möglich.
- Anheuerung von Freiwilligen ohne Budget möglich.

Flexibilität

- Keine Flexibilität, da auch kein Budget gegeben.

Veränderlichkeit

- Das Studiums-Projekt ist nicht kommerziell, daher wird sich das Budget nicht ändern.

O3. Größe der Gruppe

Auswirkungen

- Eine größere Gruppe kann theoretisch mehr Arbeit verrichten und somit eine komplexere Architektur umsetzen.
- Bei zu wenig Gruppenmitgliedern kann sich der Abschluss der Implementierung verzögern oder es werden nur Teile dieser fertiggestellt.

Flexibilität

- Die höchstmögliche Anzahl der Gruppenmitglieder steht fest und ist weder vor noch während der Arbeit verhandelbar.
- Fallen jedoch Gruppenmitglieder weg, können eventuell neue Mitarbeiter hinzukommen.

Veränderlichkeit

- Es können noch Gruppenmitglieder während der Planung oder Implementierung wegfallen oder bis zum Maximum an Gruppenmitgliedern hinzukommen.

O4. Erfahrung der Gruppenmitglieder

Auswirkungen

- Softwarearchitektur richtet sich auch nach der Erfahrung der Gruppenmitglieder mit dem Aufbau von Softwarearchitektur und zu verwendender Software.
- Bei zu wenig Erfahrung können mögliche Produkteigenschaften nicht allzu gut ausfallen oder die Impementierung verzögert sich.

Flexibilität

- Zu verwendende Software steht fest. Man kann allerdings wahrscheinlich bei Problemen auf andere Open-Source-Software umsteigen, sofern in der Anforderungsspezifikation nichts anderes vereinbart ist.

Veränderlichkeit

- Wenig Veränderung erwartet. Die Erfahrung der Gruppenmitglieder steigt zwar bei der Einarbeitung, jedoch ist dies das erste Projekt.

O5. Entwicklungsumgebung

Auswirkungen

- Zeitverlust in jedem Falle, ein Scheitern des Projektes ist auch bei einem Wechsel möglich. Fehlerhaftes Endprodukt möglich.

Flexibilität

- Die vorgegebene Entwicklungsumgebung ist Eclipse und bestimmt damit wie die Implementierung formatiert ist.
- Eclipse soll bei der Planung bedacht werden, die Gruppe könnte aber auch beim Anfang der Implementierung auf andere Tools ausweichen.

Veränderlichkeit

- Zwar ist Eclipse im Zusammenspiel mit Maven vorgegeben, sollten gravierende Probleme mit dieser Software aber die Fertigstellung des Projektes behindern, kann über einen Wechsel auf andere IDE diskutiert werden. Sollte es keine andere Alternative geben, ist der Wechsel zwingend.

O6. Systembau (Build)

Auswirkungen

- Maven soll als Build-Umgebung im Zusammenspiel mit Eclipse für das Projekt genutzt werden. Es bestimmt den strukturellen Aufbau der Projektdateien und muss bei der Implementierung beachtet werden.
- Geringer Zeitverlust, da Eclipse immernoch benutzt werden kann.

Flexibilität

- Maven ist zwar vorgegeben, jedoch kann ein Projekt auch ohne dieses zu einem ausführbaren Endprodukt geführt werden.

Veränderlichkeit

- Sollten Probleme mit XML etc. auftreten, ist der Wechsel nach den Erfahrungen von Projekten vorheriger Semester nicht unüblich.

O7. Tests

Auswirkungen

- JUnit 4.0 soll verwendet werden.
- JUnit-Tests sind vom Aufbau übersichtlich (siehe Wiki), damit sollte es zu wenig Problemen kommen.
- Probleme mit JUnit können zu unvollständigen oder fehlerhaften Tests in der Abgabe führen, jedoch wird die Fertigstellung des Haupt-Projektes nur minimal beinflusst.

Flexibilität

- Die erstellten Tests sind abgaberelevant, damit bleibt wenig Flexibilität, auch wegen der sich wenig unterscheidenden TestUnit-Alternativen.

Veränderlichkeit

- Kaum Veränderung erwartet, da JUnit einigermaßen simpel ist.

2.1.2 Technische Faktoren

author: Hubert Estenfelder

T1. Hardware - Smartphone

Auswirkungen

- Da die Gruppenmitglieder wenig Erfahrung mit der Programmierung für Endgeräte mit eher moderater Leistung (Smartphones, Tablets) haben, ist eine langsamere Laufzeit wegen unoptimierten Algorithmen möglich.
- Die Implementierung sollte wenigstens Brute-Force-Lösungen umgehen, um eine gute Benutzbarkeit auf allen Endgeräten zu garantieren.

Flexibilität

- Keine Flexibilität. Die Mindestanforderungen schreiben die Implementierung für die Android-Smartphone-App vor.

Veränderlichkeit

- Keine Veränderlichkeit wegen der Mindestanforderungen.

T2. Betriebssystem - Android,Windows,Linux
Auswirkungen • Die Implementierung muss auf allen vorgegebenen Betriebssystemen laufen. Dies stellt durch Java weniger ein Problem dar, Kompatibilitäts-Probleme sind aber dennoch nicht auszuschließen. • Kompatibilitäts-Probleme können die Fertigstellung, die Produkteigenschaften sowie die Funktionsbreite des Produktes behindern.
Flexibilität • Keine Flexibilität. Nach den Mindestanforderungen muss die Implementierung der App auf Android und die Implementierung der Web-Applikation auf Windows und Linux laufen.
Veränderlichkeit • Geringe Veränderlichkeit durch Wegfall von Gruppenmitgliedern bei Grenzfällen. (Läuft auf einem Windows-Rechner, aber nicht auf dem anderen Windows-Rechner)

T3. Implementierungssprache
Auswirkungen • Da Java verwendet werden soll, müssen die Entwickler auch mit den Eigenheiten dieser Sprache zurechtkommen.
Flexibilität • Keine Flexibilität. Die Implementierungssprache Java ist durch die Mindestanforderungen gegeben.
Veränderlichkeit • Das Produkt soll in Browsern und auf Android laufen, daher ist es auch von Entwicklerseite eher unwarscheinlich von Java zu wechseln.

T4. Datenbank

Auswirkungen

- Für die serverseite Persistenz soll eine relationale Datenbank verwendet werden.
- Das Persistenz-Framework ist frei wählbar, jedoch müssen SQL-ähnliche Abfragen verwendet werden.
- Falls das Framework oder die Datenbank gewechselt werden muss, ist die zeitliche Verzögerung der Implementierung von der Kapselung der Architektur abhängig. Je weniger Module geändert werden müssen, desto besser.

Flexibilität

- Moderate Flexibilität. Es ist den Entwicklern überlassen welche Datenbank und welches Framework dafür benutzt wird, jedoch sind nach den Mindestanforderungen SQL-ähnliche Abfragen verlangt.

Veränderlichkeit

- Veränderlichkeit ist einigermaßen warscheinlich, abhängig davon, wie gut das Framework aus der Architektur zu der dann vorhandenen Implementierung passt.

2.1.3 Produktfaktoren

author: Tamim Wahage

P1. Weitere Medientypen und Erweiterung der Attribute

Auswirkungen

- Das System sollte darauf vorbereitet sein, weitere Medientypen neben Büchern aufzunehmen (DVDs, Zeitschriften, Dissertationen). Erweiterbarkeit wird somit vorausgesetzt.
- Es sollten weitere Attribute den Medientypen hinzugefügt werden können. Dies Anpassung der Typen sollte mit geringfügigen Änderungen möglich sein.

Flexibilität

- Die Erweiterbarkeit ist keine Mindestanforderung, wäre aber dennoch in einem guten Bibliothekssystem zu empfehlen.

Veränderlichkeit

- Diese Anforderung könnte mit der Zeit als Mindestanforderung hinzugenommen werden. Die Wahrscheinlichkeit hierfür ist jedoch gering. Aufgrund der geringen Priorität besteht eine beliebige Veränderbarkeit dieser Anforderung.

P2. Benutzergruppen und -rechte

Auswirkungen

- Das System sollte zwischen mehreren Benutzergruppen unterscheiden (Leiher, Bibliothekare, Admins, Gäste)
- Das System sollte die Einhaltung der unterschiedlichen Rechte der Gruppe gewährleisten. Ein Gast sollte beispielsweise keine Bewertung durchführen können.
- Der Einfluss dieses Faktors auf die Architektur ist nach dem Erfahrungsstand der Gruppe nicht klar einzuschätzen. Wir ziehen diesen Einflussfaktor sicherheitshalber in Betracht.

Flexibilität

- Benutzer mit unterschiedlichen Rechten sind Teil der Mindestanforderung. Es gibt somit keine Flexibilität.

Veränderlichkeit

- Die Rechte der einzelnen Benutzergruppen könnte sich mit der Zeit ändern. Dementsprechend muss das System mit geringfügigen Änderungen an die neue Rechtevergabe angepasst werden können.

P3. Mehrsprachigkeit

Auswirkungen

- Das System sollte mehrere Sprachen zur Verfügung stellen und leicht zwischen ihnen wechseln.
- Als Mindestanforderung müssen mindestens zwei Sprachen zur Verfügung gestellt werden.
- Das System sollte mit einfachen Methoden auch in der Wartungsphase um weitere Sprachen erweitert werden können.

Flexibilität

- Mehrsprachigkeit ist Teil der Mindestanforderung. Wir können die Anzahl der Sprachen selber wählen. Es müssen jedoch mindestens zwei sein (Englisch, Deutsch).

Veränderlichkeit

- Die Veränderlichkeit ist gering. Möglicherweise könnten die Anzahl der Sprachen, die mindestens gefordert sind, erhöht werden.

P4. Datenbankschnittstelle

Auswirkungen

- Die Anwendung auf dem Server sollte auf die Daten innerhalb der Datenbank lesend und schreibend zugreifen können.
- Hierfür wird eine Schnittstelle zur verwendeten relationalen Datenbank benötigt.
- Hierfür kann ein Persistenz-Framework wie JPA verwendet werden *(siehe Faktor T4)*

Flexibilität

- Eine Datenbankschnittstelle ist aufgrund der Verwendung einer Datenbank (für die Medien/Benutzerdaten Speicherung) notwendig. Wir sind in der Wahl dieser Schnittstelle flexibel.

Veränderlichkeit

- Möglicherweise könnten weitere Einschränkungen zur Verwendung einer speziellen Datenbank wie MySQL oder eines speziellen Persistenz-Frameworks hinzukommen. Die Wahrscheinlichkeit ist eher gering.

P5. Client-Server Schnittstelle

Auswirkungen

- Die Website/App sollte in der Lage sein mithilfe einer Schnittstelle über das Internet Daten vom Server anzufordern
- Die Website/App sollte in der Lage sein mithilfe einer Schnittstelle über das Internet Daten auf dem Server zu ändern.

Flexibilität

- Eine Client-Server Schnittstelle zwischen App/Website und Server ist notwendig. Die genaue Spezifizierung dieser Schnittstelle ist frei wählbar.

Veränderlichkeit

- Vielleicht könnten in Zukunft genauere Anforderungen an die Client-Server Schnittstelle gestellt werden. Eine solche wäre beispielsweise, dass die Verbindung zwischen Client und Server verschlüsselt sein soll. Solche Veränderungsmöglichkeiten sollten bei der Wahl der Schnittstelle in Betracht gezogen werden.

P6. Effiziente GUI-Gestaltung

Auswirkungen

- Die App/Website sollte eine ansprechende GUI besitzen um dem Nutzer eine angenehmere und einfachere Benutzung zu ermöglichen.
- Es sollten bestimmte Designregeln der GUI-Entwicklung angewandt werden.
- Die GUI sollte weitgehend unabhängig von der Geschäftslogik laufen. Hierfür bietet sich die Verwendung bestimmter Design-Pattern an (MVC)

Flexibilität

- Die Entwicklung der GUI bietet uns großen Freiraum, da keine genauen Anforderungen an diese gestellt sind. Es sollte jedoch ein Mindestmaß an Benutzerfreundlichkeit erreicht werden.

Veränderlichkeit

- Vielleicht könnten in Zukunft genauere Anforderungen an die GUI eintreffen. In diesem Fall sollte die GUI leicht anzupassen sein.

P7. Performanz

Auswirkungen

- Alle Anwendungen (App,Website,Server Anwendung) sollten für alle Aktionen möglichst kurze Ausführungszeiten besitzen.
- Lange Ausführungszeiten sorgt beim Nutzer für Unzufriedenheit und ist ein Merkmal der Ineffizienz der Anwendung.
- Es sollten effiziente Algorithmen verwendet und unnötige Datenbankzugriffe vermieden werden. Unnötiger, redundanter Code sollte vermieden werden.
- Der Speicherbedarf der App sollte berücksichtigt werden. Aufwendige Prozesse auf einem Smartphone sind ineffizient.
- Der Start der App-Anwendung sollte nicht allzu aufwendig sein, so dass dieser kurz ist. Das gleiche gilt für die Beendigung der App.
- Die Server-Anwendung sollte auch bei einer großen Anzahl an Client-Anfragen performant bleiben.
- Bei der Wahl der Architektur sollte die Performanz berücksichtigt werden.

Flexibilität

- Es gibt keine genauen Anforderungen zu den Ausführungszeiten der Anwendung. Ein Mindestmaß für die Performanz sollte dennoch eingehalten werden.

Veränderlichkeit

- Die Performanz kann sich im Laufe der Implementierungsphase mehrmals ändern und man muss möglicherweise Kompromisse mit anderen Einflussfaktoren eingehen.

P8. Verlässlichkeit

Auswirkungen

- Die Server-Anwendung sollte die meiste Zeit verfügbar sein und nicht allzu oft abstürzen.
- Die meisten Aktionen der Anwendungen sollten fehlerfrei laufen.
- Die Datensicherheit sollte gewährleistet sein. Hierzu sollte darauf geachtet werden, dass nicht durch Fehler im System das Risiko eines Datenverlusts besteht.
- Ausgiebiges Testen sollte angewandt werden.
- Sicherheitslücken bei der Wahl der Architektur sollten berücksichtigt werden (SQL-Injection etc.)

Flexibilität

- Es gibt keine genauen Anforderungen zur Verlässlichkeit der Anwendung. Ein Mindestmaß für die Verlässlichkeit sollte dennoch eingehalten werden.

Veränderlichkeit

- Die Anforderungen an die Verlässlichkeit können sich im Laufe der Zeit verändern.

P9. Fehlererkennung und -behandlung

Auswirkungen

- Die Anwendungen sollten bei Fehlern möglichst nicht Abstürzen. Dies setzt ein effektives Exception-Handling voraus.
- Nach Auftreten des Fehlers, sollte dieser protokolliert werden und der weitere Verlauf der Anwendung gewährleistet werden.

Flexibilität

- Es gibt keine genauen Anforderungen zur Fehlerbehandlung der Anwendung. Ein Mindestmaß für die Fehlertoleranz sollte dennoch eingehalten werden.

Veränderlichkeit

- Die Anforderungen an die Fehlerbehandlung können sich im Laufe der Zeit verändern.

2.2 Probleme und Strategien

author: Alex Mues, Hubert Estenfelder, Tamim Wahage

1. Zeitprobleme

Es besteht das Problem des Zeitmangels aufgrund der Gruppengröße und/oder des Einhaltens der Deadline. Aufgrund der geringen Erfahrung der Gruppenmitglieder wird zudem mehr Zeit benötigt.

Einflussfaktoren

- *O1. Time-To-Market*
- *O3. Größe der Gruppe*
- *O4. Erfahrung der Gruppe*
- *O5. Entwicklungsumgebung*
- *O6. Systembau(Build)*

Lösung:

- **Strategie S1 : Einfache Architektur**
  Einfacher Aufbau der Architektur. Es sollte keine komplexe Architektur gewählt werden, welche unsere unerfahrenen Gruppenmitglieder überfordert.
- **Strategie S2 : Mindestanforderung**
  Das Einhalten der Mindestanforderungen hat höchste Priorität. Alle optionalen Anforderungen werden erst am Ende berücksichtigt.
- **Strategie S3 : Modularisierung**
  Eine effektive Einteilung in Module hat den Vorteil, dass man arbeitsteilig vorgehen kann. Viele Optionale Anforderungen können später aufgrund der einfachen Erweiterbarkeit durch Modularisierung hinzugefügt werden.

Alle erwähnten Strategien erscheinen sinnvoll und werden umgesetzt, zumal sie sich nicht gegenseitig ausschließen.

Verwandte Strategien:

- Keine

2. Verwendung von unbekannter Technologie

Die Gruppe hat bisher keine Erfahrung mit der Programmierung auf der ADT-Umgebung, sowie mit Maven,SQL und JPA. Die Entwicklung einer Java-Server Anwendung ist für alle Gruppenmitglieder ein neues Gebiet.

Einflussfaktoren

- *O2. Budget*
- *O4. Erfahrung der Gruppe*
- *O5. Entwicklungsumgebung*
- *O6. Systembau(Build)*
- *T1. Hardware(Smartphone)*
- *T2. Betriebssystem - Android, Windows, Linux*
- *T4. Datenbank*
- *P4. Datenbankschnittstelle*
- *P5. Client-Server Schnittstelle*

Lösung:

- **Strategie S4 : Schnittstellen zu Technologien**
  Wir versuchen unser Projekt mithilfe der Modularisierung unabhängig von einer bestimmten Technologie zu entwickeln. Hierzu werden die Technologien abgekapselt und über bestimmte Schnittstellen angesprochen. Die Schnittstellen sind so implementiert, dass die Technologie durch eine leichtere Alternative ersetzt werden kann (z.B. SQLite anstelle von MySQL).
- **Strategie S5 : Implementierung von eigener Technologie**
  Wir implementieren eigene Technologien anstatt bestehende Frameworks etc. zu verwenden. Zum Beispiel wird eine eigene Persistenz-Schnittstelle implementiert, anstatt JPA zu verwenden.

Wir entscheiden uns in diesem Fall für die Strategie S4, da diese viel Flexibilität bezüglich der Wahl von Technologie- Alternativen bietet. Die Strategie S5 ist zu aufwendig für Technologien wie SQL und erfordert zu viel Zeit. Somit verzichten wir auf diese.

Verwandte Strategien:

- **S4 :** Ist mit S3 aus dem Problem 1. verwandt.

3. Benutzbarkeit der GUI

Die GUI sollte benutzerfreundlich sein. Sie sollte bei Aktionen möglichst nicht einfrieren. Die Benutzerschnittstelle sollte intuitiv bedienbar sein und nicht fehleranfällig sein.

Einflussfaktoren

- *O1. Time-To-Market*
- *O3. Größe der Gruppe*
- *O4. Erfahrung der Gruppe*
- *P2. Benutzergruppen und -rechte*
- *P3. Mehrsprachigkeit*
- *P6. Effiziente GUI-Gestaltung*
- *P7. Performanz*
- *P8. Verlässlichkeit*
- *P9. Fehlererkennung und -behandlung*

Lösung:

- **Strategie S6 : Verwenden von Design-Pattern**
  Für die Entwicklung der GUI auf der App und der Website versuchen wir auf bewährte Design Pattern wie MVC (Model-View-Controller) zurückzugreifen, welche eine effiziente Entwicklung der GUI ermöglicht.

Wir verwenden S6, da wir keine ähnlich einfach aufgebaute Strategie gefunden haben.

Verwandte Strategien:

- **S6 :** Ist mit S3 aus 1. und mit S4 aus 2. verwandt.

4. Änderung der Anforderung

Mit der Zeit könnten Anforderungen hinzukommen. Es könnte Schwierigkeiten geben beim fortgeschrittenem Projekt diese noch umzusetzen. Umgekehrt könnten Anforderungen auch wegfallen. Es könnte eventuell Probleme geben die betroffenen Funktionen zu entfernen ohne die System Integrität zu gefährden.

Einflussfaktoren

- *O1. Time-To-Market*
- *O2. Budget*
- *O3. Größe der Gruppe*
- *P1. Weitere Medientypen und Erweiterung der Attribute*
- *P2. Benutzergruppen und -rechte*
- *P3. Mehrsprachigkeit*
- *P4. Datenbankschnittstelle*
- *P5. Client-Server Schnittstelle*
- *P6. Effiziente GUI-Gestaltung*
- *P7. Performanz*
- *P8. Verlässlichkeit*
- *P9. Fehlererkennung und -behandlung*

Lösung:

- **Strategie S7 : Abkapselung der Mindestanforderung**
  Durch Modularisierung der einzelnen Anforderungen sorgen wir für eine gewisse gegenseitige Unabhängigkeit der Anforderungen zueinander.
- **Strategie S8 : Erweiterbarkeit**
  Die Module und Schnittstellen sollten durch neue Module erweiterbar sein.

In Hinsicht auf SWP2 wählen wir als Strategie S8 um neue Anforderungen zum implementierten Prototypen im Blockkurs hinzufügen zu können. Es ist unwahrscheinlich, dass während der Implementierungsphase implementierte Mindestanforderungen wegfallen, so dass wir auf S7 vermutlich verzichten können.

Verwandte Strategien:

- **S7 :** Ist mit S2 und S3 aus 1. verwandt.
- **S8 :** Ist mit S3 aus 1. verwandt.

5. Mehrsprachigkeit

Das System könnte aufgrund der Möglichkeit des Wechsels der Sprache aufwendig implementiert werden. Beschriftungen müssen an vielen Stellen umständlich angepasst werden. Der Bibliothekar muss Medienbeschreibung in mehreren Sprachen angeben. Hinzufügen von weiteren Sprachen könnte Probleme bereiten.

Einflussfaktoren

- *P3. Mehrsprachigkeit*
- *P6. Effiziente GUI-Gestaltung*

Lösung:

- **Strategie S9 : Abstrakte Bezeichner und lokale Sprachdateien**
  Für die Beschriftungen der GUI werden abstrakte Bezeichner verwendet. Jedem dieser Bezeichner wird für jede Sprache eine entsprechende Beschriftung zugeordnet. Dies geschieht über ein zusätzliches Modul. Beim Schalten der Sprache werden die Beschriftungen angepasst. Die Beschriftungen sind in lokalen Sprachdateien auf dem Client verfügbar.
- **Strategie S10 : Abstrakte Bezeichner und zentrale Sprachdateien**
  Ähnlich wie in S9 werden abstrakte Bezeichner, wobei jedoch die Sprachen nicht lokal in Sprachdateien gespeichert sind sondern auf dem Server. Auf dem Server können die Sprachen auch in der Datenbank gespeichert werden.
- **Strategie S11 : Mehrere Versionen der Anwendung**
  Die Beschriftungen sind fest eingestellt. Für jede Sprache wird eine Version der Anwendung mit unterschiedlichen Beschriftungen kompiliert.

Wir schätzen S11 als ineffizient ein. Wir werden eine Kombination aus S9 und S10 verwenden. Die Sprachdateien sollten sowohl auf dem Client als auch auf dem Server vorhanden sein.

Verwandte Strategien:

- Keine

6. Verfügbarkeit des Servers

Die Server-Anwendung sollte mit zahlreichen Anfragen von Clients zurecht kommen. Es besteht das Risiko, dass der Server bei zu vielen Anfragen zusammenbricht und nicht mehr verfügbar ist. Ist die Schnittstelle zwischen Server und Client fehlerhaft, so kann es auch bei einzelnen Anfragen zu Problemen kommen.

Einflussfaktoren

- *O4. Erfahrung der Gruppenmitglieder*
- *T2. Betriebssystem - Android, Windows, Linux*
- *T4. Datenbank*
- *P4. Datenbankschnittstelle*
- *P5. Client-Server Schnittstelle*
- *P7. Performanz*
- *P8. Verlässlichkeit*
- *P9. Fehlererkennung und -behandlung*

Lösung:

- **Strategie S12 : Nebenläufigkeit**
  Wir verwenden für die Abfertigung der Client-Abfragen mehrere Threads.
- **Strategie S13 : Effiziente Datenbankabfragen**
  Wir verwenden schlanke Datenbankabfragen.
- **Strategie S14 : Effiziente Schnittstellenimplementierung zwischen Client und Server** Wir legen Wert auf eine effiziente Schnittstelle zwischen Client und Server.

Wir würden vorzugsweise alle Strategien umsetzen.

Verwandte Strategien:

- Keine

7. Performanz

Es besteht das Risiko von zu langen Ausführungszeiten. Das Starten der App, sowie das Beenden könnte zu lange dauern. Bestimmte Prozesse auf der App könnten hinsichtlich der geringen Leistung eines Smartphones Probleme verursachen. Ineffiziente Algorithmen und Redundanter Code auf Server- und Clientseite könnten unnötige Wartezeiten verursachen.

Einflussfaktoren

- *O4. Erfahrung der Gruppenmitglieder*
- *T1. Hardware(Smartphone)*
- *T2. Betriebssystem - Android, Windows, Linux*
- *T3. Implementierungssprache*
- *T4. Datenbank*
- *P4. Datenbankschnittstelle*
- *P5. Client-Server Schnittstelle*
- *P7. Performanz*
- *P8. Verlässlichkeit*

Lösung:

- **Strategie S15 : Effiziente Algorithmen**
  Wir verwenden effiziente Algorithmen mit guten Laufzeiten.
- **Strategie S16 : Auswahl einer schnellen Datenbank**
  Wir wählen eine schnelle Datenbank aus, die für, die von uns geschätzte Datenmenge gute Zugriffszeiten ermöglicht.

Wir verwenden alle Strategien.

Verwandte Strategien:

- Keine

8. Datensicherheit

In unserer Anwendung könnten Sicherheitslücken auftreten. Fehlerhafte Funktionen könnten versehentlich Daten aus der Datenbank löschen. Benutzer könnten fremde Daten einsehen.

Einflussfaktoren

- *O4. Erfahrung der Gruppenmitglieder*
- *T4. Datenbank*
- *P2. Benutzergruppen und -rechte*
- *P4. Datenbankschnittstelle*
- *P5. Client-Server Schnittstelle*
- *P9. Fehlererkennung und -behandlung*

Lösung:

- **Strategie S17 : Überprüfen von Benutzereingaben**
  Durch das Überprüfen von Benutzereingaben sollte SQL-Injection verhindert werden.
- **Strategie S18 : Ausgiebiges Testen**
  Wir wählen eine Architektur aus, die uns das Testen von einzelnen Modulen oder ganzen Komponenten erleichtert.

Wir verwenden alle Strategien.

Verwandte Strategien:

- Keine

9. Schlechte Arbeitsteilung

Bei schlechter Koordination der einzelnen Arbeitspakete könnten inkompatible Komponenten der Anwendung entstehen, die gemeinsam gar nicht oder nur eingeschränkt laufen.

Einflussfaktoren

- *O3. Größe der Gruppe*
- *O4. Erfahrung der Gruppenmitglieder*
- *P4. Datenbankschnittstelle*
- *P5. Client-Server Schnittstelle*

Lösung:

- **Strategie S19 : Spezifizieren von einheitlichen Schnittstellen**
  Wir versuchen für alle Komponenten und Module einheitliche Schnittstellen zu verwenden.

Wir verwenden alle Strategien.

Verwandte Strategien:

- **S19 :** Ist verwandt mit S14 aus 6. , S4 aus 2. und S3 aus 1.

10. Benutzerauthentifikation

Ein Benutzer sollte nur Aktionen entsprechend der eigenen Benutzerrechte durchführen können. Ein Leiher sollte beispielsweise nicht in der Lage sein Ausleihen durchzuführen. Die Anmeldung eines Benutzers sollte beim Server und/oder Client gespeichert werden.

Einflussfaktoren

- *P2. Benutzergruppen und -rechte*
- *P5. Client-Server Schnittstelle*

Lösung:

- **Strategie S20 : Verwenden von Sessions**
  Bei der Anmeldung erhält der Client ein Session-Objekt mit einem generierten Schlüssel und seinen Benutzerdaten. Eine Kopie des Objekts wird auch beim Server gespeichert.

Wir verwenden alle Strategien.

Verwandte Strategien:

- Keine

3 Konzeptionelle Sicht

author: Hubert Estenfelder

Für den groben Architetkturstil verwenden wir das Model-View-Controller-Pattern:

- **Persistence:**
  Die Komponente Persistence ist die Schnittstelle zur Datenbank und setzt Abfragen der Server-Logic in sprechende Datenbankabfragen um.

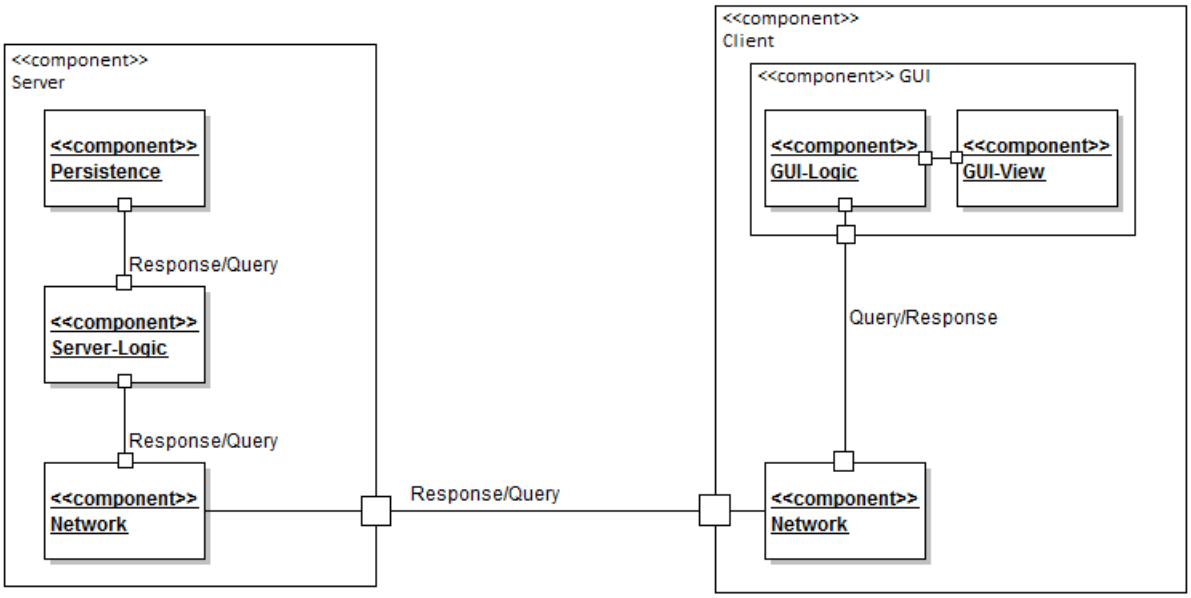


Abbildung 1: Komponenten-Diagramm

- **Server-Logic:**
  Die Server-Logic setzt Anfragen des Networks in Befehle der Persistence um. Gleichzeitig hält sie alle bei ihr angemeldeten *Observer* (hier GUI) und informiert sie über Änderungen der Datenbank. Sie übernimmt die Geschäftslogik des Servers und gehört zum Teil zum Model.
- **Network:**
  Nimmt Anfragen der GUI-Logic auf der Client-Seite entgegen und schickt sie an den Server. Genauso werden Nachrichten des Servers vom Network an den Clienten und dessen GUI gesendet.
- **GUI-Logic:**
  Die GUI-Logic übernimmt die Rolle des Controllers und nimmt Eingaben des Nutzers sowie Nachrichten des Networks entgegen und übergibt sie dem GUI-View.
- **GUI-View:**
  Übernimmt die Rolle des Views und zeigt dem Benutzer die gewünschten Daten an.

4 Modulsicht

4.1 Paketdiagramm

author: Alex Mues, Hubert Estenfelder, Tamim Wahage

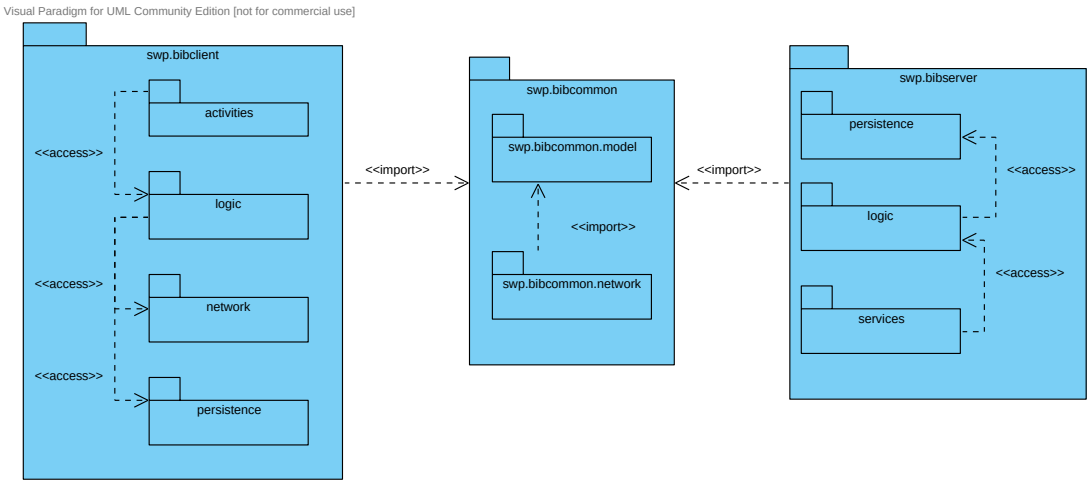


Abbildung 2: Paket-Diagramm

4.2 Module

4.2.1 Common

author: Alex Mues

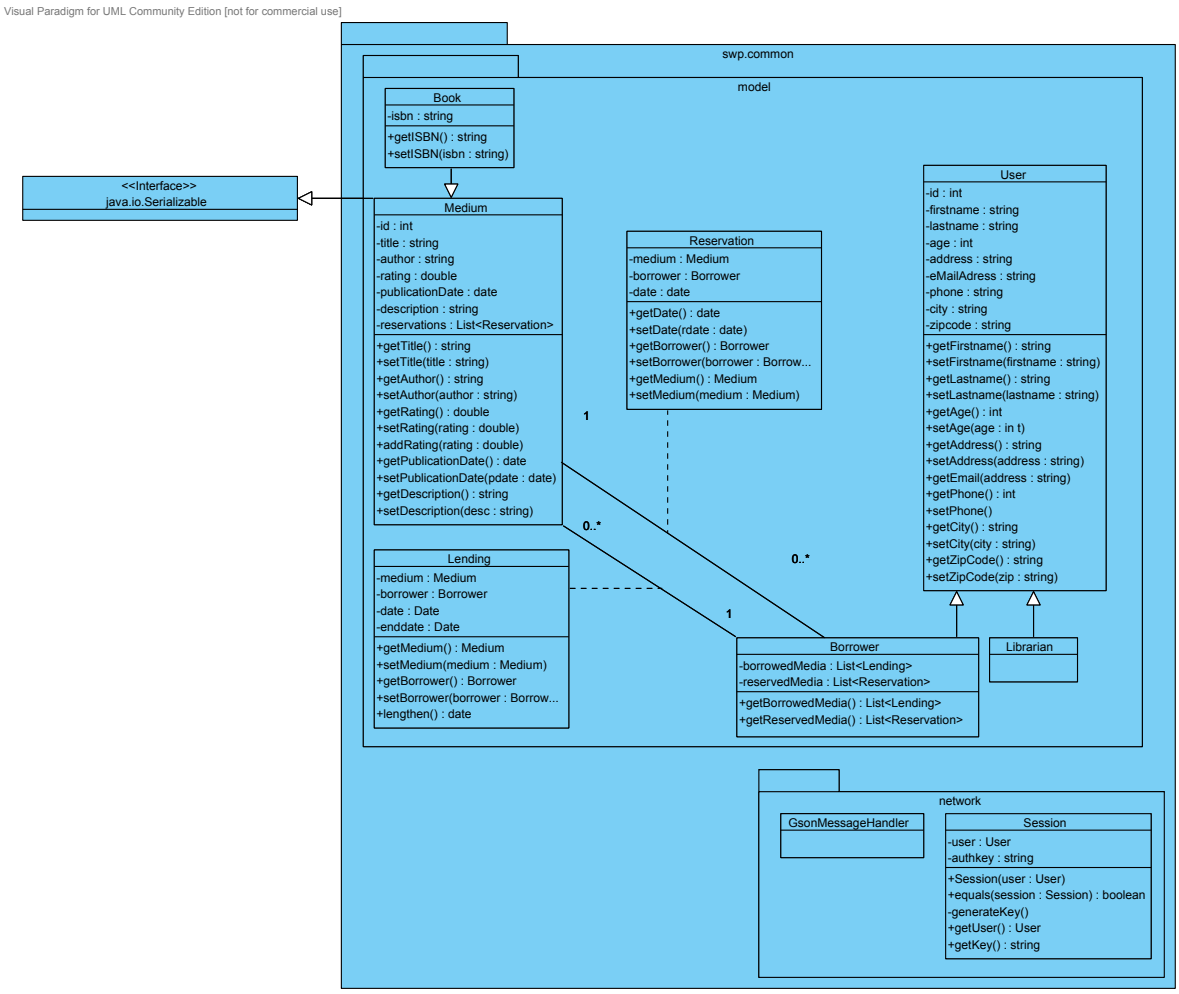


Abbildung 3: Common-Modul

Das *swp.bibcommon*-Paket enthält Klassen und Unterpakete die sowohl vom Client-Modul als auch vom Server-Modul verwendet werden.

model Hier befinden sich die Klassen unseres Datenmodells, wie *Medium* oder *User*. Objekte dieser Klassen werden in einer Datenbank persistiert. Eine nähere Beschreibung der einzelnen Klassen befindet sich in der Datensicht.

network Hier befinden sich die Klassen die von der Client-Server-Schnittstelle verwendet werden. Hierzu gehören der "GsonMessageHandler" und die "Session" Klasse. Der "GsonMessageHandler" wird für die Übertragung der Objekte im JSON-Format verwendet. Die "Session"-Klasse wird für die Authentifizierung von Usern verwendet. Eine genauere Beschreibung befindet sich beim Server-Modul bei der Beschreibung des Session-Managers.

4.2.2 Server

author: Tamim Wahage Das *swp.bibserver*-Paket entspricht der Server-Komponente aus der konzeptionellen Sicht. Folgende Pakete sind im *swp.bibserver*-Paket enthalten.

services Dieses Paket entspricht der *Network*-Schnittstelle des Servers in der konzeptionellen Sicht. Die Klasse "BibServices" stellt mithilfe des Jersey-Frameworks nach dem *JAX-RS* Standard REST-Dienste zur Verfügung, die vom App-Client verwendet werden können. Hierfür wird das Jersey-Servlet *com.sun.jersey.spi.container.servlet.Servlet .Container* verwendet. "BibServices" kann auf den "BusinessHandler" und somit direkt auf die Logik zugreifen. Für den Zugriff des Web-Clients auf den Server verwenden wir *Java-Server-Faces(JSF)*. Hierzu integrieren wir das Servlet *javax.faces.webapp.FacesServlet*. Über xhtml-Dateien kann der Web-Client auf die *Java Managed Beans* zugreifen, die im *services* Paket bereitgestellt werden. Diese bieten Dienste für den Web-Client an und greifen wie die "BibServices"-Klasse direkt auf den "BusinessHandler" zu. Das Servlet "BibServlet" dient der Konfiguration des Systems (z.B. initialisieren des Loggings. Bisher sind keine Konfigurationen für das Servlet geplant, was sich im Laufe der Implementierung ändern kann. Die Verbindung zwischen Client und Server sollte SSL verschlüsselt werden.

logic Dieses Paket entspricht der *Server-Logic*-Komponente des Servers aus der konzeptionellen Sicht. Sie enthält alle Klassen die für die Geschäftslogik des Servers verantwortlich sind. Bisher sind die beiden Klassen "BusinessHandler" und "SessionManager" enthalten. Bei beiden handelt es sich um *Singletons*. Der "BusinessHandler" bietet den Diensten aus dem *services*-Paket entsprechende Methoden an. Zum Beispiel wird in der "BibServices"-Klasse der Dienst *getAllBooks()* angeboten. Entsprechend gibt es im "BusinessHandler" eine Methode *getAllBooks()*, die von dem REST-Dienst aufgerufen wird. Der "BusinessHandler" kann direkt auf die Persistenz-Schnittstelle zugreifen um in diesem Beispiel aus der Datenbank alle Bücher zu erhalten.
Der "SessionManager" verwaltet alle laufenden *Sessions* der angemeldeten Benutzer. Ein Benutzer kann sich beim System anmelden und erhält eine Session mit einem generierten Schlüssel. Bei der Anmeldung erhält der "SessionManager" eine Kopie der Session. Die Session enthält neben dem generierten Schlüssel auch das *User*-Objekt des angemeldeten Benutzers. Bei bestimmten Aktionen, wie z.B. dem Hinzufügen eines Leihers

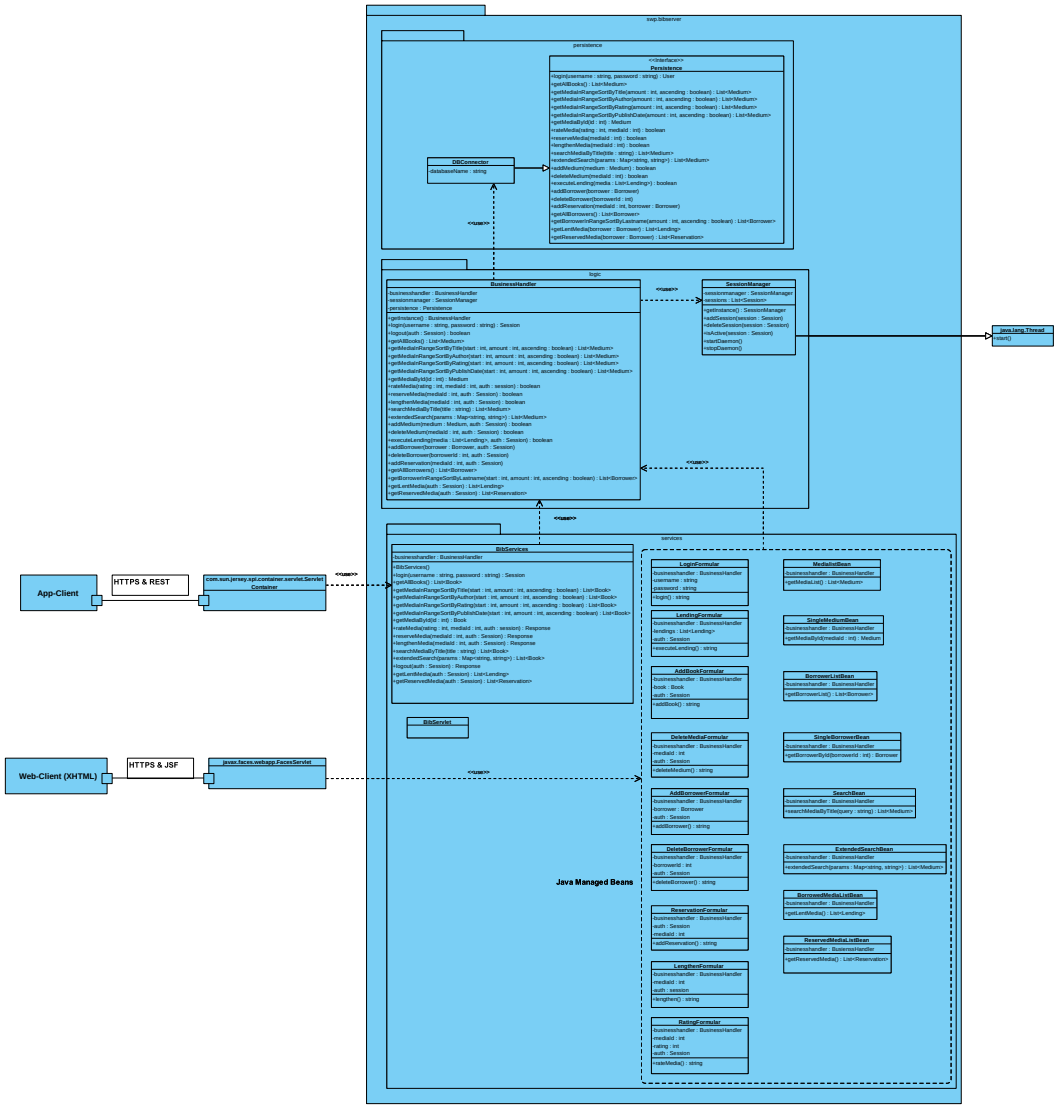


Abbildung 4: Server-Modul

wird das *Session*-Objekt vom Client mit übergeben. Der "BusinessHandler" überprüft mithilfe des enthaltenen *User*-Objekts, ob es sich beim Benutzer um einen Bibliothekar handelt, der dazu berechtigt ist Leiher hinzuzufügen. Gleichzeitig überprüft er auch beim "SessionManager" ob die Session noch aktiv ist. Ist dies nicht der Fall, so muss sich der Client erneut anmelden. Der "SessionManager" löscht einzelne Sessions nach einer bestimmten Zeitspanne, wenn der User lange Zeit keine Aktionen mehr ausführt. Die Session kann auch manuell gelöscht werden indem sich der User gezielt abmeldet.

persistence Dieses Paket entspricht der *Persistenz*-Komponente des Servers aus der konzeptionellen Sicht. Sie enthält alle Klassen die für das Persistieren von Daten und Objekten in Datenbanken und/oder Dateien zuständig sind und aus der Datenbank und/oder den Dateien Daten abfragen können. Innerhalb des Pakets wird ein Interface "Persistence" definiert. Das Interface gibt Methoden für die Persistierung vor, die jede Klasse dieses Pakets implementieren muss. Zum Beispiel muss jede Persistenzklasse die Methode *getAllBooks()* zum Erhalten aller Bücher implementieren. Die Klasse "DB-Connector" realisiert die "Persistenz"-Schnittstelle und dient als Schnittstelle zu einer relationalen Datenbank. Sie bietet dem "BusinessHandler" Methoden zur Datenhaltung und -abfrage an.

verwendete Strategien

- Wir haben für den Server eine vergleichsweise simple Architektur gewählt und damit Strategie **S1** umgesetzt.
- Aufgrund der Zeitprobleme wurden lediglich die Mindestanforderungen umgesetzt. Für unsere 3-Mann Gruppe wurde die Mindestanforderungen "*Hinzufügen und Löschen von Bibliothekare*" entfernt. Hiermit wurde die Strategie **S2** umgesetzt.
- Wir haben einen modularen Aufbau unserer Systems spezifiziert und somit Strategie **S3** umgesetzt.
- Wir definieren ein *Persistence*-Interface und ermöglichen damit eine Unabhängigkeit und Austauschbarkeit der verwendeten Datenbank. Hiermit wurde Strategie **S4** umgesetzt.
- Wir verwenden beim Server eine leicht abgeänderte Form des MVC-Pattern. Der View befindet sich beim Client und die Netzwerkschnittstelle mit REST funktioniert wie ein verlängerter Arm der Client-Logik, welche auf die Server-Logik zugreift. Hiermit wurde Strategie **S6** umgesetzt.
- Wir verwenden für die Server-Client-Schnittstelle bewährte Standards wie REST für den App-Client und JSF für den Webclient. Zudem wird mittels *HTTPS* die Verbindung mit *SSL* verschlüsselt. Hiermit wird die Strategie **S14** umgesetzt.
- Die von uns gewählte Architektur und der modulare Aufbau ermöglichen effizientes Testen. Hiermit wurde Strategie **S18** umgesetzt.
- Es werden bei den Diensten, beim "BusinessHandler" und bei der Persistenz-

Schnittstelle einheitliche Methodennamen verwendet. Hiermit wird Strategie **S19** umgesetzt.

- Für die Benutzerauthentifikation verwenden wir *Sessions*. Hiermit wird Strategie **S20** umgesetzt.

4.2.3 Client

author: Hubert Estenfelder

Für die Android-App wird folgender Aufbau verwendet:

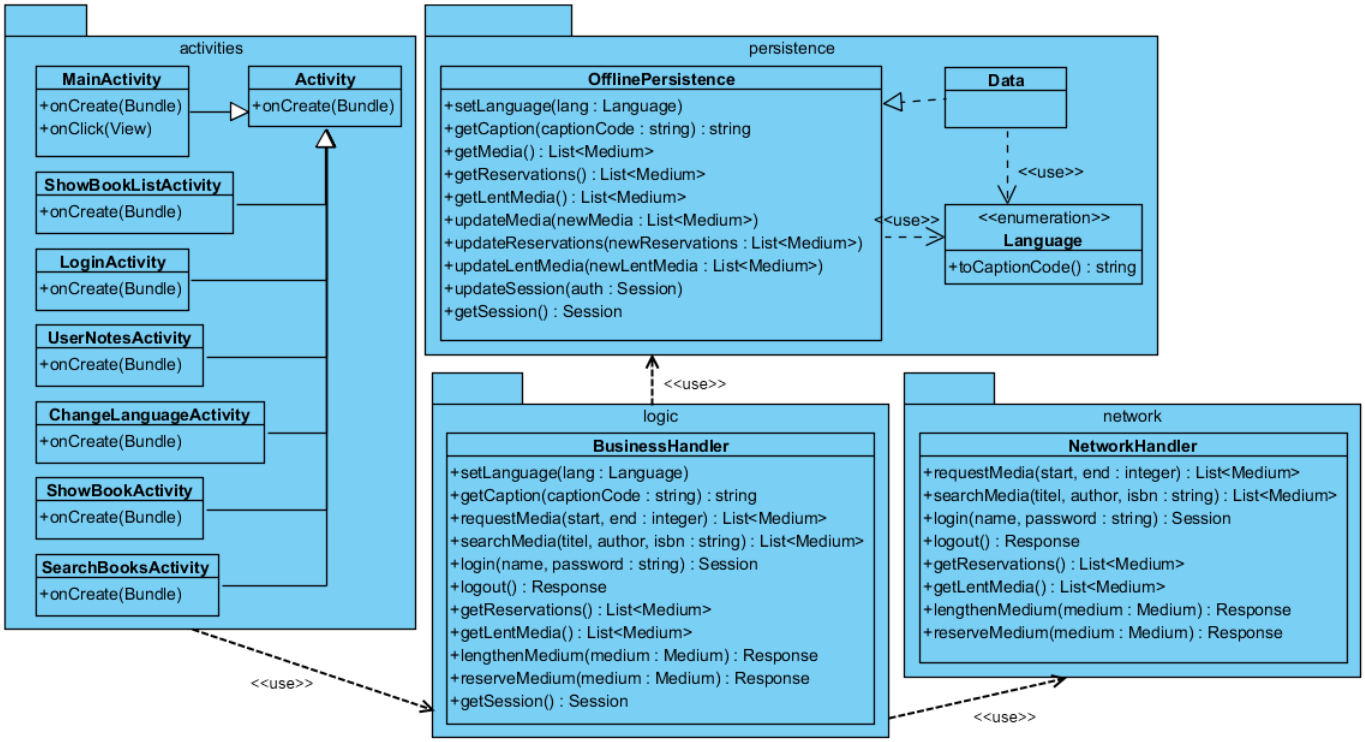


Abbildung 5: Client-Modulsicht

Folgende Pakete sind in dem *bibclient*-Paket enthalten:

- *activities* Die Sammlung aller Activities der Android-App. Sie bedienen sich der Methoden, die der BusinessHandler bereitstellt. "activities" beschreibt die GUI-View-Komponente, ist also Ein-und Ausgabe.

- *persistence* Die "OfflinePersistence" bildet ein Interface, um Daten auf dem Android-Gerät abzurufen und temporär abzulegen. Es ist für die Ablage der Session und das Laden aller Beschriftungen für die versch. Sprachen zuständig. Im Sinne der Komponentendarstellung ist dies Teil der GUI-Logik.

- *network* Dieses Paket besteht nur aus der "NetworkHandler"-Klasse, die für das Aufbereiten und Verschicken von Nachrichten des Clients zuständig ist. In der Komponentendarstellung übernimmt network die gleich-benannte Komponente, da sie als Schnittstelle zum Server fungiert.

- *logic* Dieses Paket besteht nur aus einem Singleton, dem "BusinessHandler", er sammelt die Methoden von NetworkHandler und der Persistenz Data und stellt sie den Activities zur Verfügung, die auf die anderen Instanzen keinen Zugriff haben sollen. Der BusinessHandler verteilt diese Aufrufe dann entsprechend an das Persistenz- und das Netzwerk-Paket. Dieses Paket übernimmt die Funktion der GUI-Logik in der Komponentendarstellung.

Paket: *activities*
Die verschiedenen Activities benötigen alle die Methode onCreate(), um ihren vorherigen Stand aus dem Hintergrund wiederherzustellen. Der Großteil der eigentlichen Anwendungsfälle wird hier privat gelöst.Dieses Paket bildet die GUI.

Paket: *persistence*
Das Interface OfflinePersistence wird komplett ohne Veränderungen von Data implementiert. Die einzelnen Methoden sind hier kurz erläutert:

- *setLanguage()*: Setzt die Sprache entsprechend einem Parameter aus der Enumeration Language. Von nun an werden für die Caption-Codes (z.B. "SEARCH_BUTTON") die für die Sprache entsprechenden Beschriftungen an die Activities gegeben.

- *getCaption()*: Gibt für einen Caption-Code (s.o.) die für die Sprache entsprechende Beschriftung zurück.

- *getMedia()*: Gibt die zuletzt vom Server gelieferte Liste an Medien zurück.

- *getReservations()*: Gibt die zuletzt vom Server gelieferte Liste an vorgemerkten Medien zurück.

- *getLentMedia()*: Gibt die zuletzt vom Server gelieferte Liste an ausgeliehenen Büchern zurück.

- *updateMedia()*: Aktualisiert die Liste von Medien.

- *updateReservations()*: Aktualisiert die Liste von vorgemerkten Medien.

- *updateLentMedia()*: Aktualisiert die Liste von ausgeliehenen Medien.
- *updateSession()*: Aktualisiert die Sitzungsdaten.
- *getSession()*: Gibt die Daten der aktuell aktiven Session zurück.

Paket: *network*
Die Klasse NetworkHandler verschickt Anfragen des Nutzers und empfängt die entsprechenden Daten von Server. Die einzelnen Methoden sind hier kurz erläutert (Die natürlich nur korrekte Rückgaben liefern, wenn der Nutzer auch angemeldet ist):

- *requestMedia()*: Gibt einen Bereich in der Datenbank aller Medien an, von dem die Medien abgefragt werden sollen. In welcher Ordung sie angespochen und zurückgeliefert werden, ist nicht genauer spezifizert und kann variabel sein.
- *searchMedia()*: Gibt Medien vom Server zurück, die den angegebenen Suchkriterien entsprechen.
- *login()*: Nimmt den Benutzernamen und das Passwort eines Nutzers und meldet ihn beim Server an. Zurückgegeben wird eine Instanz von Session, die die Daten zu dieser Sitzung enthält.
- *logout()*: Meldet den Benutzer von der Verbindung zum Server ab.
- *getReservations()*: Fordert alle vorgemerkten Medien dieses Nutzers vom Server.
- *getLentMedia()*: Fordert alle ausgeliehenen Medien dieses Nutzers vom Server.
- *lengthenMedium()*: Verlängert die Ausleihe eines bestimmten Mediums, sofern es möglich ist.
- *reserveMedium()*: Fügt der Liste an vorgemerkten Medien ein bestimmtes Medium hinzu. Es ist nicht möglich die Vormerkung aufzuheben, sie wird nach Verfall der Vormerkdauer aufgehoben.

Paket: *logic*
Der im Paket logic vorhandene BusinessHandler stellt die gleichen Methoden wie Data und NetworkHandler den Activities zur Verfügung. Da also die Methoden in diesen Klassen schon erläutert wurden, wird an dieser Stelle darauf verzichtet, da die Bennenungen eindeutig sind. Der BusinessHandler muss ein Singleton sein, da alle

Activities auf ein und dieselbe Geschäftslogik zugreifen sollen. Außerdem hält er die Instanzen von NetworkHandler und Data.

verwendete Strategien:

- Auch hier wurde mit einer möglichst einfachen Architektur die Strategie S1 übernommen.
- Folgender Anwendungsfall wurde vom Android-Client entfernt: Passwort per Email anfordern. Damit wurde Strategie S2 umgesetzt.
- Strategie S3 wurde in dem Sinne umgesetzt, das Eingabe/Ausgabe im Paket *activities*, die Geschäftslogik in *logic*, die Netzwerk-Schnittstelle in *network* und die Offline-Persistenz im Paket *persistence* gekapselt sind.
- Strategie S4 wurde mithilfe des Pakets *activities* umgesetzt, da hier alle android-relevanten Klassen eingeschlossen sind, die nur mit dem BusinessHandler interagieren.
- Das MVC-Pattern aus Strategie S6 ist teilweise durch die Ähnlichkeit der Android-App zu einem Viewer umgesetzt. Jedoch sendet die App anfragen und der Server antwortet nur. Dies ist so, um ständige Aktualisierungen während der Benutztung zu unterbinden.
- Da die App vollständig vom Server abgekapselt ist und sich nur der von BibServices angebotenen Funktionen bedient, können einfach neue Module hinzugefügt werden und nur mit geringen Änderungen der Schnittstellen angepasst werden.
- Für die Spracheinstellungsmöglichkeiten aus den Mindestanforderungen haben wir uns für die Strategie S9 entschieden. Die Sprachdateien werden von der client-seitigen Persistenz verwaltet und sind über den BusinessHandler des Clienten verfügbar.
- Strategie S19 spiegelt sich in der hohen Ähnlichkeit der Aufrufe innerhalb des Clients wieder und genauso in der vom NetworkHandler verwendeten Funktionen des Server wieder (einheitliche Schnittstellen).
- Strategie S20 wurde durch das Annehmen und speichern von einer Session in der client-seitigen Persistenz übernommen.

5 Datensicht

author: Alex Mues

Unsere Datensicht entspricht im wesentlichen der Modulsicht zu bibcommon.

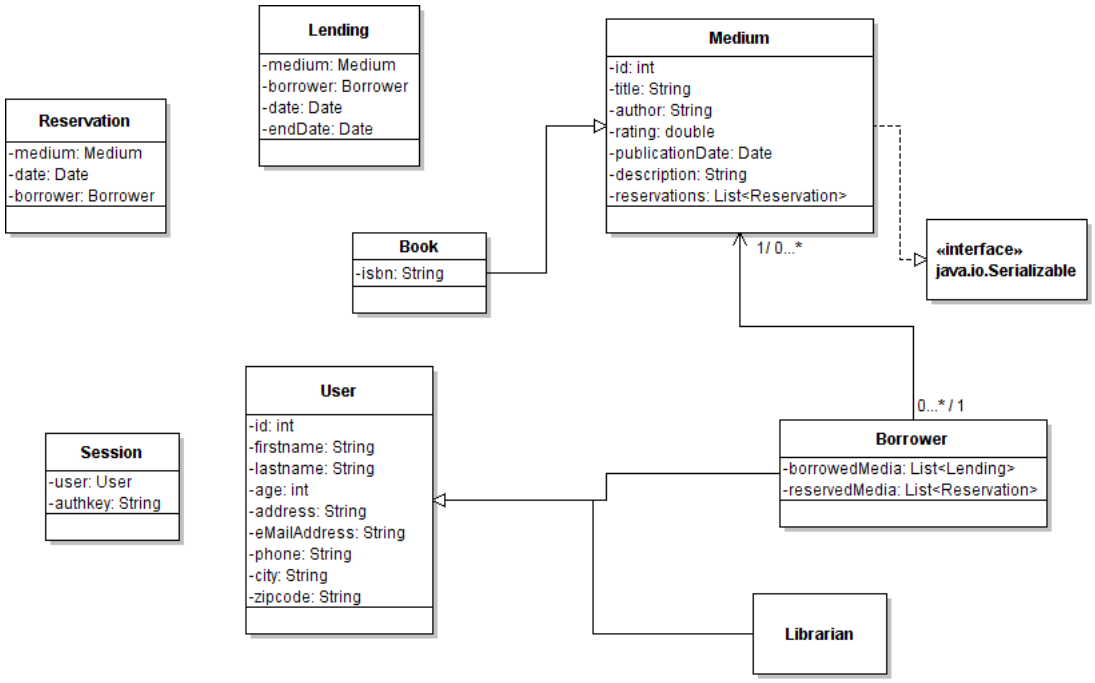


Abbildung 6: Datensicht

Es gibt eine Klasse Medium, die die Superklasse von allen zur Verfügung stehenden Leihobjekten der Bibliothek darstellt. Da in dem momentanen Bibliotheksprojekt nur Bücher ausgeliehen werden können, ist sie vor allem für spätere Erweiterungen gedacht. Sie besitzt alle Datentyp, die ein Leihobjekt einer Bibliothek besitzen muss.
Jedes Medium der Bibliothek hat eine ID. Jedes Medium hat außerdem einen Titel, einen Autor, eine textuelle Beschreibung und ein Publikationsdatum.
Darüber hinaus ist es möglich das Medium zu bewerten, weshalb Medium eine Bewertung hat. Wird ein neues Medium erzeugt, liegt die Bewertung standardmäßig bei 0, da noch keine Bewertung abgegeben wurde.
Medium hat außerdem eine Liste, die Reservations-Objekte speichert. Dies ist notwendig, da es für einen Leiher(Borrower) möglich ist, ein Medium zu reservieren. Medium implementiert außerdem das Serializable Interface. Ihre Objekte können also serialisiert werden.
Die Klasse Book erbt von Medium. Da sie momentan die einzige Klasse ist, die von Medium erbt, ist sie das einzige Medium der Bibliothek. Zusätzlich zu den anderen Attributen eines Mediums, hat Book außerdem noch eine ISBN.
Leiher(Borrower) und Bibliothekare(Librarian) sind User.
Ein User hat eine eigene ID, einen Vor- bzw. Nachnamen, ein Alter, eine Adresse, eine E-Mail Adresse, eine Telefonnummer eine Heimatstadt und eine Postleitzahl. Das Datenfeld für das Alter wird für Projekte implementiert, die eventuell auf diesem aufbauen. Es wird eigentlich nur gebraucht, wenn es Leihobjekte mit einer gewissen Altersbeschränkung gibt. Ein User besitzt zudem auch einen Benutzernamen und ein Passwort für die Anmeldung beim System. Diese werden jedoch nicht im Objekt

gespeichert, sondern befinden sich aus Sicherheitsgründen nur in der Datenbank.
Die Klasse Borrower wird außerdem noch um eine Liste mit den bestehenden Ausleihen erweitert. Die Liste enthält Referenzen auf Lending Objekte. Diese stellen wiederum Ausleihen dar (Medium, Buch, Zeitpunkt etc.). Das heißt, dass jeder Ausleiher eine Liste mit seinen Ausleihen hat. Darüber hinaus haben Borrower Objekte eine Liste, die Referenzen auf Reservation Instanzen enthält. Sie speichert also die Reservationen, die der Ausleiher getätigt hat.
Lending hat Datenfelder für ein Medium, einen Ausleiher, ein Ausleihdatum und ein Enddatum, an dem das Medium zurückgegeben werden muss. Da ein Borrower für jedes ausgeliehene Medium eine Referenz auf eine Lending Instanz in seiner borrowedMedia Liste hält, sind alle für die jeweilige Ausleihe relevanten Daten im Borrower selbst enthalten.
Jeder Leiher kann beliebig viele Medien ausleihen. Für jedes ausgeliehene Medium gibt es eine eigene Lending.
Es ist außerdem möglich, dass beliebig viele Borrower das gleiche Medium reservieren. Sie reservieren dasselbe Medium nacheinander, sodass das Reservieren desselben Mediums von verschiedenen Leihern für die selbe Zeitspanne nicht möglich ist.
Reservierungen werden durch Objekte der Reservation Klasse dargestellt. Reservation enthält also alle Daten, die für eine Reservierung relevant sind. Dazu gehört das Medium, das reserviert wird, der Borrower, der reserviert hat, und das entsprechende Datum.
Jeder User bekommt bei der Verbindung mit dem Server eine Referenz auf ein neues Session Objekt. Die Session Klasse enthält Datenfelder für den User, dem das Session Objekt zugewiesen wurde und einen Authentifizierungsschlüssel. Es befinden sich identische Session Objekte auf Server und Client Seite. Wenn man die Verbindung trennt, muss sich der User neu anmelden, und erhält somit eine neue Session.

6 Ausführungssicht

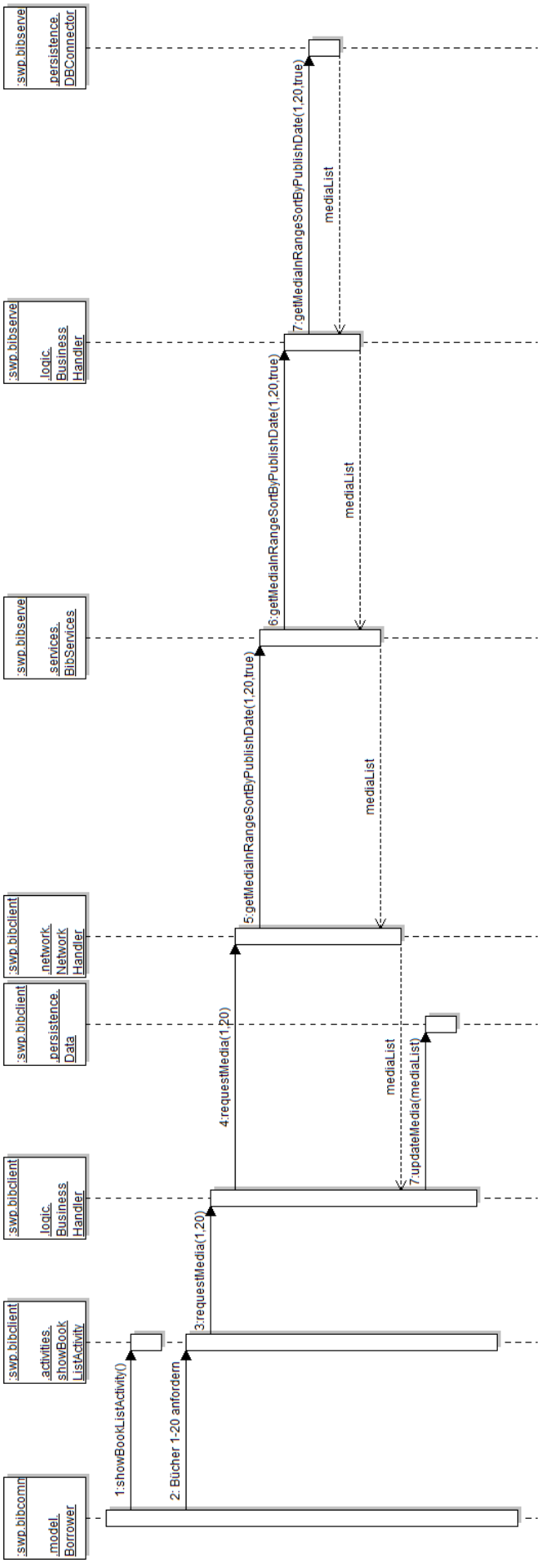
Entfällt in SWP-1

7 Zusammenhänge zwischen Anwendungsfällen und Architektur

Für den Anwendungsfall wird eine der grundlegensten Operationen ausgewählt, die ein Benutzer ausführen kann: Er lässt sich die neuesten 20 Medien in der normalen Listensicht ausgeben.
Vorbedingungen: Der Benutzer hat die Android-App erfolgreich gestartet.
Vorgehen: Der Benutzer startet eine ShowBookListActivity und fordert damit automatisch den Anfangsteil aller neuhinzugekommenen Medien. In diesem Fall ist die Liste

an Medien auf der ersten Listenseite 20 Medien lang. Diese Forderung wird über den BusinessHandler des Clients, den NetworkHandler, die BibServices-Klasse des Server, den BusinessHandler des Servers und dessen Persistenzklasse DBConnector.

Daraus ergibt sich folgendes Sequenzdiagramm:


:swp.bibcomm
model.
Borrower
:swp.bibclient
activities.
showBook
ListActivity
:swp.bibclient
logic.
Business
Handler
:swp.bibclient
persistence.
Data
:swp.bibclient
network.
Network
Handler
:swp.bibserve
services.
BibServices
:swp.bibserve
logic.
Business
Handler
:swp.bibserve
persistence.
DBConnector
1:showBookListActivity()
2: Bücher 1-20 anfordern
3:requestMedia(1,20)
4:requestMedia(1,20)
5:getMediaInRangeSortByPublishDate(1,20,true)
6:getMediaInRangeSortByPublishDate(1,20,true)
7:getMediaInRangeSortByPublishDate(1,20,true)
mediaList
mediaList
mediaList
mediaList
7:updateMedia(mediaList)

8 Evolution

Entfällt in SWP-1